LDSI W2021 Project Report

Name: Siyu Zhou
Gloss ID: tum_ldsi_9

Summary

In this project, we first compared the performance of the sentence segmenter provided by spaCy and a Law-specific sentence segmenter named LUIMA SBD in segmenting legal documents, and the better performing LUIMA SBD was used to segment 30,000 unlabeled legal decisions. Then, based on the segmented sentences, we trained a word embedding model using FastText. At last, with the word embedding model and TFIDF featurization, we trained classifiers based on different machine learning algorithms and discuss their results in depth.

Corpora Description

In this project, we used an annotated legal decision corpus and an unannotated corpus both from the US Board of Veterans Appeals (BVA). The annotated corpus consists of 70 granted and 71 denied decisions used to train and test our final classifier. The unannotated one contains 30000 legal cases, which will be used to train our word embedding model.

Dataset Splitting

We split the annotated corpus into three balanced sub-datasets, respectively 80% as the training set (56 granted 57 denied cases), 10% as the test set (7 granted and 7 denied cases), and 10% as the development ("dev set") set (7 granted and 7 denied cases). The case IDs of the test and dev set are as below. Then the remaining documents form the training set.

Dev set:
['61aea55e97ad59b4cfc412f0', '61aea55d97ad59b4cfc412c7', '61aea55d97ad59b4cfc412cd', '61aea55d97ad59b4cfc412b7', '61aea55c97ad59b4cfc412aa', '61aea55e97ad59b4cfc412d8', '61aea55c97ad59b4cfc412a4', '61aea57397ad59b4cfc41399', '61aea57197ad59b4cfc4136b', '61aea56f97ad59b4cfc4134c', '61aea57097ad59b4cfc41364', '61aea57397ad59b4cfc4139c', '61aea57297ad59b4cfc41380', '61aea57097ad59b4cfc41355']

Test set:
['61aea55f97ad59b4cfc41334', '61aea55d97ad59b4cfc412d2', '61aea55f97ad59b4cfc41337', '61aea55f97ad59b4cfc41319', '61aea55d97ad59b4cfc412d3', '61aea55f97ad59b4cfc41328', '61aea55f97ad59b4cfc41308', '61aea57497ad59b4cfc413e3', '61aea57197ad59b4cfc41375', '61aea57497ad59b4cfc413e7', '61aea57497ad59b4cfc413c0', '61aea57197ad59b4cfc41376', '61aea57497ad59b4cfc413d8', '61aea57497ad59b4cfc413b3']

Sentence Segmentation

Our final classifier can only accept and process a representation of a specific sentence. Therefore, we are supposed to segment all the documents in the corpus into sentences. To do this, we have two options. The first one is to use the segmenter of spaCy, and another one is to use a law-specific sentence segmenter named LUIMA SBD. This section will apply both methods to our training set to figure out which option is optimal.

- Default spaCy segmenter
  The default spaCy sentence segmenter did not perform well on our training set, with a precision of 0.602, recall of 0.620, and f1-score of 0.611. Then we examined three documents with low document-level precision/recall values manually, to find some

common mis-segmentation patterns. We selected documents “1615107.txt”, “1231338.txt”, and “1550735.txt” and found some common over- and under-segmented instances.

1. Over-segmentation for “Vet. App.”
   We observed in some sentences within the text “Vet. App.”, the segmenter may over-split the sentence based on the periods of “Vet. App.”. As the figure is shown below, the segmenter mis-split the Citation sentence “See Pond v. West, 12 Vet. App. 341 (1999); Macarubbo v. Gober, 10 Vet. App. 388 (1997); Caluza v. Brown, 7 Vet. App. 498 (1995); Cartright v. Derwinski, 2 Vet. App. 24 (1991).” into five sub-sentences.

```
-------------------------------------------------
 See Pond v. West, 12 Vet. App.
-------------------------------------------------
341 (1999); Macarubbo v. Gober, 10 Vet. App.
-------------------------------------------------
388 (1997); Caluza v. Brown, 7 Vet. App.
-------------------------------------------------
498 (1995); Cartright v. Derwinski, 2 Vet. App.
-------------------------------------------------
24 (1991).
-------------------------------------------------
```

Figure 1: Example of over-segmentation for “Vet. App.”

2. Over-segmentation for “No.”
   For the opening CaseHeader sentence of every document, the segmenter over-segment it based on the period of “No.”. An example is shown below.

```
Citation Nr: 1550735
Decision Date: 12/03/15    Archive Date: 12/10/15

DOCKET NO.
-------------------------------------------------
 14-35 196       )       DATE
         )
         )
```

Figure 2: Example of over-segmentation for “No.”

3. Under-segmentation for the opening texts
   We observed the segmenter always under-segment the opening texts of every document. The figure below shows an example generated by the segmenter. However, according to our annotation standard, segmentation should be done between the opening CaseHeader, the Procedure sentence “On appeal from the …” and the Header sentence “THE ISSUE”.

```
Citation Nr: 1550735
Decision Date: 12/03/15    Archive Date: 12/10/15

DOCKET NO.
-----------------------------------------------------------------------------------------
 14-35 196      )       DATE
        )
        )

On appeal from the
Department of Veterans Affairs Regional Office in San Juan, the Commonwealth of Puerto Rico


THE ISSUE

Entitlement to service connection for bilateral hearing loss.
-----------------------------------------------------------------------------------------
```

Figure 3: Example of under-segmentation for the opening texts

4. Under-segmentation for Header
   We found the segmenter has difficulty in segmenting the Header sentences. As the example shown below, the segmenter failed to segment the Header sentence “FINDING OF FACT”.

```
FINDING OF FACT

The competent medical evidence does not demonstrate that the Veteran's currently diagnosed bilateral hammertoes are a
ttributable to his active duty service or any incident of service, to include as secondary to a service-connected dis
ability.
```

Figure 4: Example of under-segmentation for Header

- Improved spaCy segmenter
  According to the error analysis of the default spaCy sentence segmenter in the last section, we added the following exceptions/extensions to improve it and applied the improved segmenter on the training data again.
  1. Making the “Vet. App.” a single token.
     We added a special case to the tokenization pattern of spaCy, making spaCy regard the “Vet. App.” as a single token, so as to make the segmenter ignore the periods in “Vet. App.”.
  2. Making the “No.” a single token.
     We added another special case similarly, making spaCy regard the “No.” as a single token, so as to make the segmenter also ignore the period in “No.”.
  3. Separating sentences by multiple newline characters.
     After observation on the characteristics of Header sentences, we found nearly every header is preceded by three lines and followed by two lines. Thus, we overwrote the configuration of spaCy’s sentence segmenter (Sentencizer), making the segmenter regard the “\r\n\r\n\r\n” and “\r\n\r\n” also as sentence boundaries like period.

  At last, the improved segmenter performed much better with a precision of 0.813, recall of 0.885, and f1-score of 0.847.

- LUIMA SBD
  In this section, we applied Savelka's law-specific sentence segmenter LUIMA SBD on the training data similarly. It outperformed the default spaCy segmenter and also the improved one. We got a result of precision of 0.831, recall of 0.990, and f1-score of 0.904. Then we also delve deeper into the actual segmentation result and found the LUIMA SBD over-segment sentences more frequently than spaCy segmenter. Here are three over-segmentation error examples of LUIMA.
  1. The following example shows how the LUIMA SBD segmenter over-segments an opening CaseHeader sentence based on its line breaks.

```
Citation Nr: 1538335
---------------------------------Segmentation Line-------------------------------------
Decision Date: 09/08/15
---------------------------------Segmentation Line-------------------------------------
Archive Date: 09/18/15
---------------------------------Segmentation Line-------------------------------------
DOCKET NO.  06-05 125
---------------------------------Segmentation Line-------------------------------------
DATE
---------------------------------Segmentation Line-------------------------------------
```

Figure 5: Example 1 of segmentation error of LUIMA SBD

  2. In the following example, the LUIMA SBD segmenter over-segments the sentence “II.Legal Criteria and Analysis” into two parts based on the period following the Roman number.

```
--------------------------------Segmentation Line-------------------------------------
II.
--------------------------------Segmentation Line-------------------------------------
Legal Criteria and Analysis
--------------------------------Segmentation Line-------------------------------------
```

Figure 6: Example 2 of segmentation error of LUIMA SBD

3. As above, the LUIMA SBD segmenter over-segments the sentence “1.The Veteran participated in a radiation risk activity during service.” due to the period following the Arabic number.

```
--------------------------------Segmentation Line-------------------------------------
1.
--------------------------------Segmentation Line-------------------------------------
The Veteran participated in a radiation risk activity during service.
--------------------------------Segmentation Line-------------------------------------
```

Figure 7: Example 3 of segmentation error of LUIMA SBD

According to the error examples of the appeal, we found that on the one hand, the LUIMA SBD segmenter tends to treat some consecutive newlines and periods as sentence boundaries, leading to excessive segmentation. However, LUIMA SBD can recognize common patterns in legal texts that the non-legal segmenter of spaCy can not, such as “Vet. App.” And “No.”, to avoid some over-segmentations.

On the other hand, the LUIMA SBD segmenter can also avoid the most unnecessary under-segmentations of spaCy sentence segmenter, which is also why LUIMA SBD segmenter can achieve better performance on our training data.

In summary, the LUIMA SBD segmenter can outperform spaCy’s default segmenter and our improved one on legal texts, according to the precision, recall, and f1-score. Although the LUIMA SBD segmenter still faces the problem of over-segmentation, it is uncommon and non-dominant. Therefore, we decided to use the LUIMA SBD segmenter to process our unlabeled corpus in the next section.

Preprocessing

In this section, we applied the LUIMA SBD on all documents of our unlabeled corpus and tokenized all segmented sentences.

- Segmenting unannotated data
  After segmenting sentences of all unlabeled documents, we got 3488106 sentences in total. The figure below shows the number distribution of sentences across all the 30000 unlabeled legal documents. The minimum and maximum number of sentences in a document are 20 and 1039 respectively. The average number of sentences across all unlabeled documents is 116.27.

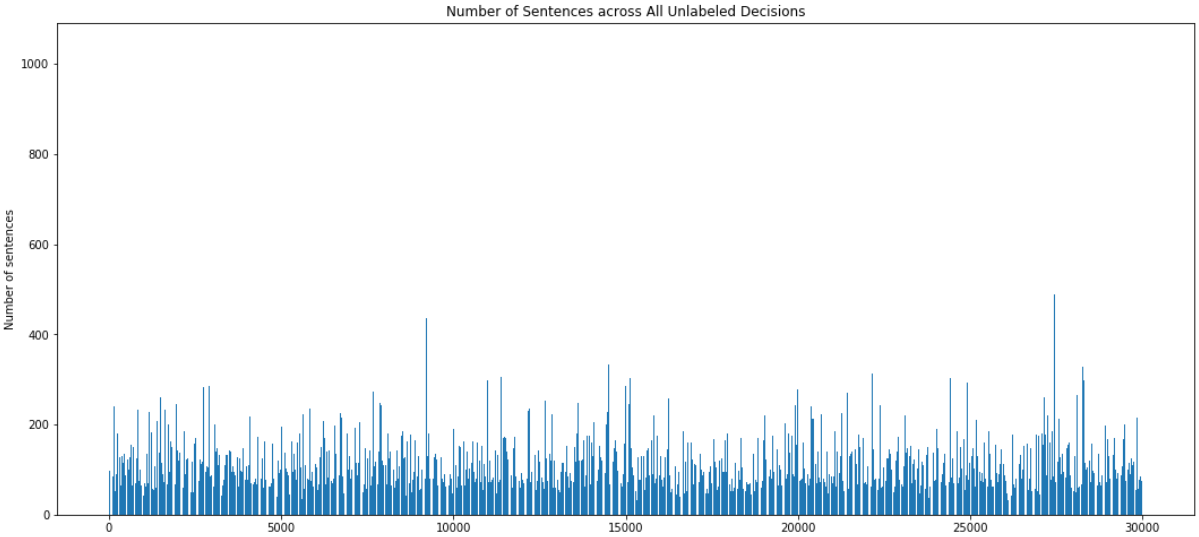

Figure 8: number distribution of sentences across all unlabeled documents

- Sentence-wise preprocessing
  In order to tokenize all sentences segmented from the unlabeled corpus, we created a tokenizer function that splits a sentence into tokens with uninformative noise reduction. To be specific, the tokenizer will:
  1. skip all punctuations and other meaningless symbols such as "/" and "-"
  2. skip single or consecutive space characters
  3. skip the newline "\r\n"
  4. skip the tab "\t"
  5. convert possessive form, e.g., from veteran's to veteran, from lawyers' to lawyer
  6. convert all numbers to label <NUM>, including numbers that contain English letters like 3.156(a)
  7. lowercase all tokens (words)

  With the steps above, our tokenizer can filter most useless tokens, including punctuations, symbols, space characters, escape characters, and different numbers, and reduce the different representations of similar tokens, including possessive and case differentiation.

  To examine our tokenizer function, we applied it to a series of legal sentences and tokenize them manually at the same time. At last, we compared the result of both, which revealed the results of our tokenizer function almost exactly matched the manual results. Therefore, we consider our tokenizer function good enough.

- Tokenize unannotated data
  With the tokenizer function of the appeal, we tokenized all sentences segmented from the unlabeled corpus. Finally, the minimum, maximum, and average numbers of tokens of a sentence are 0, 461, 17.57.

  We discarded sentences with the number of tokens under 3 and then stored all remaining sentences in a file named "unlabeled_sentences.txt" each line of which consists of a sentence's tokens, separated by a single whitespace. The sentences in the file are in random order.

Developing Word Embeddings

In this section, we made use of all the tokenized sentences generated in the last section to train a word embedding model.

- Custom FastText embeddings
  We use FastText to train the word embedding model based on the file "unlabeled_sentences.txt". The model parameter settings included: mode = "skipgram", dim = 100, minn = 2, maxn = 5, minCount = 20, epoch = 30. (For the description of model parameters, please see https://fasttext.cc/docs/en/options.html.)

- Evaluating Custom Embedding Manually
  The vocabulary size of the word embedding model is 13270. To examine the model, we explored nearest neighbors for various strings, including "veteran", "v.", "argues", "ptsd", "granted", "korea", "holding", "also", "service", "disease", and "testimony". We then found some interesting patterns from the results and will discuss them next.
  1. "veteran": The "veteran"s closest neighbor is "appellant", which makes sense because the "veterans" always plays the role of the "appellant" in decision documents of our corpus. However, the next nearest neighbors mostly have little to do with the "veteran" itself in terms of meaning, such as the, he, that, etc. The reason for this should be the word "veteran" a very high-frequency word in our corpus, which makes our model tend to regard it as a commonly-used word like "the" and "he".
  2. "v.": The string "v." usually appears in the middle of a name like "Grantham v. Brown", and has no specific meaning, which makes its nearest neighbors difficult to explain in terms of meaning. From the result, the top two neighbors are "vet." and "app.", which should be because they have a similar structure, i.e., a period followed by a word. There are also some name strings in its closest neighbors list. The reason for this should be that these people have "v." in their names.
  3. "argues": The closest neighbor of "argues" is "argue". Other neighbors are mostly close in terms of meaning, such as "appealed", "reiterate", "unequivocally", and "agree". However, there are also a few neighbors which are difficult to explain like "holbrook" (person's name), "informal", and "cox".
  4. "pstd": pstd means post-traumatic stress disorder. Its closest neighbors are mostly related to mental diseases including "depressive", "mdd" (major depressive disorder), "dysthymia", "anxiety", etc. which are reasonable in terms of meaning.
  5. "granted": the closest neighbor of "granted" is unwarranted, which is antithetical to itself. Our model should only consider the action of words without their tendencies. The second and fifth neighbors are "granting" and "grant", which is reasonable. But the remaining neighbors are mostly unexplained to some extent, including person's name, "prejudicated", etc.
  6. "korea": The closest neighbor of "korea" is "korean", which is similar to itself in terms of meaning. Besides, most of the other neighbors are also location names such as "gemany", "dmz" (Demilitarized Zone), "vietnam", etc, which are also reasonable.
  7. "holding": The closest neighbors of "holding" are mostly people's names. It is possible that these people held something in the decision documents of our corpus. However, we even did not find the string "hold" in the closest neighbors list, which is somewhat unreasonable.
  8. "also": The word "also", as a common-used word, whose closest neighbors are mostly also common words such as "see", "<NUM>", "that", and "although", which usually appear in legal texts. In addition, the words "addition", and "additionally" also belong to the closest neighbors of "also". We can interpret this to mean that they all have the meaning of a supplementary description.
  9. "private": We found the closest neighbors of "private" are mostly related to medical treatment, such as "treating", "provider", "dr.", and "record". After reviewing the BVA corpus, we found that "private" is often found in sentences related to medical diagnosis. An example is "A July 2001 private treatment report noted that he had witnessed tragic and violent deaths during his military service." Therefore, we consider the pattern should make sense.

10. “disease”: Most of the closest neighbors of the word “disease” are also highly related to disease, such as “incur”, “diseased”, “injury”, “disability”, “aggravate”, etc. Therefore, the pattern also makes sense in terms of meaning.
11. “required”: The second closest neighbor of “required” is “requisite”, which is reasonable because of the same meanings. Part of the other nearest neighbors are also adjectives such as “acceptable”, “available”, “unreasonable”, “specified”, and “arguable”. They are also similar to “required” in terms of lexical and modifying objects, and therefore reasonable. However, there also exists some unexplainable neighbors for “required” such as “avail”, “mail”, and “admonish”.

Based on the above analysis, we believe that this word embedding model basically meets the requirements, i.e., it can fit the distance between words relatively well. Although there are still some words for which we found it difficult to interpret some of their nearest neighbors, for most words, especially those with clear concepts, such as "disease", their nearest neighbors are consistent with our knowledge of the BVA corpus.

Training & Optimizing Classifiers

In this section, we trained sentence classifiers based on different machine learning algorithms and two featurization, respectively TFIDF and word embedding discussed in the last section. At first, we need to apply TFIDF and word embedding featurization on our dataset.

- TFIDF featurization
  We used the class TfidfVectorizer of sklearn to train a vectorization model based on the training data. The trained model contained 2898 features (words). Then we apply this model on our training data, dev data, and testing data, to convert all the sentences to vectors with the dimension of 2898.

- Word embedding featurization
  We leveraged the word embedding model we have trained to convert all the sentences of the training, dev, and testing set to vectors with a length of 100. Then we calculated the normalized position and the normalized number of tokens of these sentences, and append them to the corresponding word embedding vectors. As a result, the dimension of the feature vector of a sentence is 102.

After the steps above, we obtained two types of sentence feature vectors for the training, dev, and testing set. We then used both of them to train the classifier respectively.

- Classifier training
  We trained multiple models using TFIDF and word embedding featurization respectively, including SVM, Logistic Regression, Decision Tree, Random Forests, and Naive Bayes. After constant re-training with adjusted parameters and comparing different algorithmic models, we decided the best models TFIDF featurization and word embedding featurization are both Radial Basis Function (RBF) kernel SVM with default hyperparameter settings. The decision was based on the final classification metrics on training and dev set, that is, the model largely outperforms other algorithmic models to some extent in terms of weighted averaged precision (WAP), weighted averaged recall (WAR), weighted averaged f1 (WAF) and accuracy (Acc). Then we applied both of the models to the testing set. The table below shows the concrete metrics for the training, dev, and testing set.

Table 1: Classifier performance metrics on training, dev, and testing set

	Training Set				Dev Set				Testing Set			
	WAP	WAR	WAF	Acc	WAP	WAR	WAF	Acc	WAP	WAR	WAF	Acc
SVM+TFIDF	0.94	0.95	0.94	0.95	0.83	0.84	0.82	0.84	0.81	0.83	0.81	0.83
SVM+Word Embedding	0.84	0.85	0.84	0.85	0.82	0.84	0.82	0.84	0.82	0.84	0.82	0.84

From the result of the table above, we can notice that there exists overfitting in the SVM+TFIDF model because the model performed much better on the training set than the dev and testing set. In contrast, the SVM+word embedding model behaved evenly. In general, the two models can both achieve acceptable performance on the three datasets.

Error Analysis

In order to analyze the classification error in depth, we plotted the confusion matrices for the best TFIDF and word embedding classifier on the dev set as follows.

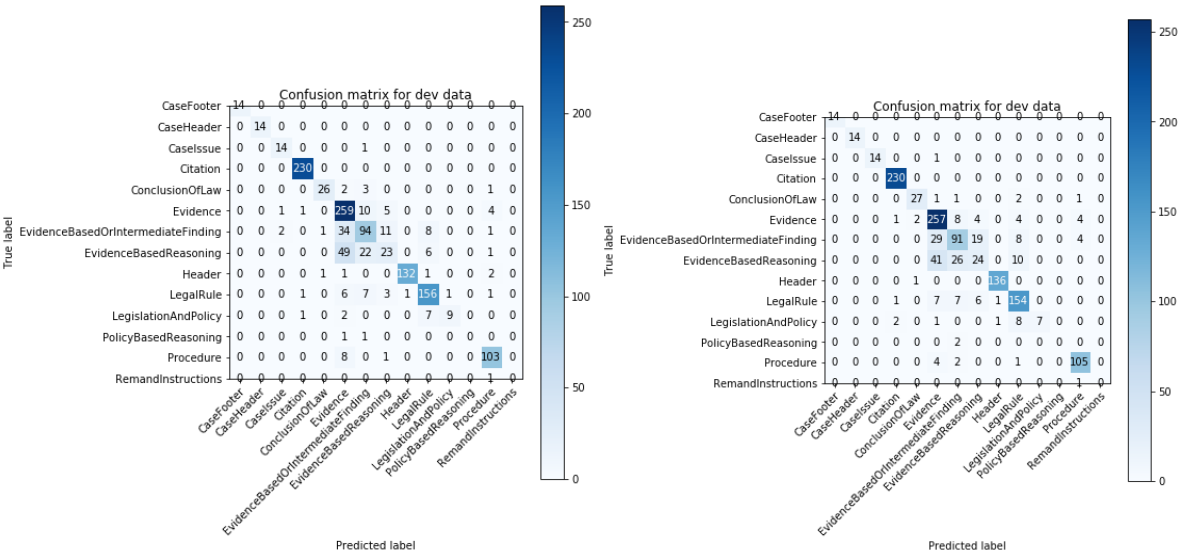

Figure 9: (a) Confusion matrix for TFIDF+SVM classifier (b) Confusion matrix for word embedding+SVM classifier

The confusion matrixes reveal that, for both classifiers, most misclassifications happen across EvidenceBasedOrIntermediateFinding, EvidenceBasedReasoning, and LegislationAndPolicy, the first two of which were often confused with each other and Evidence. The phenomenon is also common in manual annotation because of their close logical links, and relatively abstract definitions. For LegislationAndPolicy, we observed the number of misclassified samples is small, but the proportion is close to the proportion of correct classifications, so it is also a major error-prone type. Most of the other types could be identified relatively accurately, except for PolicyBasedReasoning, RemandInstructions, and Evidence.

According to the **Training & Optimizing Classifiers**, the performances of the TFIDF and word embedding classifier are very close on the dev and testing set. Therefore, we simply consider the TFIDF classifier as our overall best classifier due to its higher metrics on training data and looked at a sample of its false positive examples on the dev set for the three most difficult types as well as some other types, to find the internal factor leading to the misclassifications.

- EvidenceBasedOrIntermediateFinding

Our samples found that some false positive cases for EvidenceBasedOrIntermediateFinding were caused by wrong annotation, mainly annotated as EvidenceBasedReasoning. For instance, the EvidenceBasedOrIntermediateFinding sentence “All pertinent due process requirements have been met.” was wrongly annotated as EvidenceBasedReasoning. The relatively blurred distinction between the two types makes the error common in the annotation task.

For those false positive cases of misclassification, the reason for it should be that the sentence has certain characteristics of EvidenceBasedOrIntermediateFinding. For example, the Evidence sentence “The Board also acknowledges that the veteran has not been afforded a VA examination in connection with his claim for service connection for dental trauma” was misclassified as “EvidenceBasedOrIntermediateFinding”. The sub-sentence “The Board also acknowledges that” may make the classifier regard the whole sentence as an intermediate legal conclusion by the board, leading to misclassification as EvidenceBasedOrIntermediateFinding ultimately. Our classifier should be able to distinguish whether a sentence within a court/board as the subject expressing something is expressing an objective fact or a fact-based conclusion of the court/board.

- EvidenceBasedReasoning
  On the one hand, Similarly, some false positive cases for EvidenceBasedReasoning were also caused by the wrong annotation as EvidenceBasedOrIntermediateFinding. On the other hand, the factor for those false positive cases of misclassification should lie in the existence of some characteristics of EvidenceBasedReasoning in the sentence. For instance, the Evidence sentence “Here, the Veteran's statements are not corroborated by any medical evidence during service.” was misclassified as EvidenceBasedReseoning. Although the sentence is a declarative sentence explaining evidence, i.e., the Veteran’s statements, it is still expressing an objective fact without legal assessment. However, it is possible that the classifier regarded it as a subjective statement regarding the evidence by the board, and so classified it as EvidenceBasedReasoning. To solve this, the classifier should be improved to take words like “corroborated” into account when processing evidence-related sentences, which implies, to some extent, that the sentence is Evidence.

- LegislationAndPolicy
  We noticed the false positive cases for LegislationAndPolicy were concentrated in the samples with the true type LegalRule. After analysis one by one, we found they mostly contain time information. For instance, the LegalRule sentence “The Veterans Claims Assistance Act of 2000 (VCAA) describes VA's duty to notify and assist claimants in substantiating a claim for VA benefits.” Was misclassified as LegislationAndPolicy. Because most LegislationAndPolicy samples in our dataset contain historical information about rules, the classifier may consider the time information as an important contribution to the type LegislationAndPolicy, leading to the misclassification. To avoid it, the classifier should be further improved to pay attention to some common “prompt words” in LegislationAndPolicy sentences such as “codified” and “amended”.

- Other types
  We first explored some false positive cases for Evidence and found most of them are EvidenceBasedOrIntermediateFinding and EvidenceBasedReasoning in fact. Excluding the incorrectly annotated cases, the main reason should be the subjective inference or legal determination of these sentences is relatively implicit. Besides, We noticed the classifier did not classify any sentence as PolicyBasedReasoning or RemandInstructions, which is because of the small sample size of these two types in

the dataset used, respectively 25 and 2. In the future, more samples of these two types should be collected, so as to train a more robust model.

Discussion

In this project, we compared the sentence segmenter of spaCy and LUIMA SBD and found the LUIMA SBD segmenter can outperform spaCy's segmenter in terms of metrics because LUIMA SBD avoids most under-segmentation faced by spaCy. Then we applied the LUIMA SBD and customized tokenizer function to the unlabeled corpus, to get massive unlabeled tokenized sentences, which are then used to train the word embedding model. By observing the closest neighbors of different strings, we believe the word embedding model works well because closest neighbors of a specific string are explainable in terms of meaning, POS, and usage scenario, although there are still some closest neighbors that seem to be unreasonable to some extent, especially for words with abstractive meaning. At last, we used word embedding featurization and TFIDF featurization to train classifiers based on different machine learning algorithms. After in-depth comparison and analysis, we found the Radial Basis Function (RBF) kernel SVM model performed best both for word embedding featurization and TFIDF featurization. The TFIDF classifier had a much better performance on the training data than the word embedding classifier. However, the two classifiers behaved very close on the dev and testing data, implying the TFIDF classifier was overfitted. We also delved deeper into the classification of the TFIDF model on error-prone types by analyzing their false-positive cases, including EvidenceBasedOrIntermediateFinding, EvidenceBasedReasoning, and LegislationAndPolicy as dominant error types, as well as PolicyBasedReasoning, RemandInstructions, and Evidence as secondary error types. We found the classifier still had difficulty in distinguishing Evidence EvidenceBasedOrIntermediateFinding, and EvidenceBasedReasoning correctly, which is also common in manual annotations. Besides, the misclassification for LegislationAndPolicy was mainly because the time information in LegalRule misled the classifier. In contrast, the false positive classifications on PolicyBasedReasoning, RemandInstructions are mainly due to lack of sample.

Lessons Learned

I have learned a lot from this project. To be specific, I mastered how to convert the natural language to the vector processable by machine model, including sentence segmentation, tokenization, word embedding featurization, and TFIDF featurization. I also learned common machine learning algorithms on NLP problems, as well as domain knowledge of legal texts. As a result, my analysis ability on legal text and model improved.

From my own perspective, the most challenging parts of this project are the preprocessing and model error analysis. The former takes up a lot of computing time, and minor errors can result in much more time being invested. The latter requires me to have a good understanding of the domain knowledge of the legal text and to be able to think deeply about the inherent interpretability of the machine learning model.

Code Deliverable Description

main.ipynb: Code for all experiment steps required by the project instruction
analyze.py: Callable program of the best classifier. Given the path of a decision file, it will return a list, each of whose elements is a dictionary including keys "sentence" and "type".
word_embedding.bin: Word embedding model generated by FastText
TFIDF.pickle: TFIDF vectorizer model generated by sklearn
TFIDF_classifier.pickle: TFIDF+RBF SVM classifier model generated by sklearn
WE_classifier.pickle: word embedding classifier model generated by sklearn
luima_sbd: Source code of LUIMA SBD segmenter downloaded from GitHub